Ensembles of example-dependent cost-sensitive decision trees

April 28, 2015

Alejandro Correa Bahnsen

with

Djamila Aouada, SnT
Björn Ottersten, SnT
Motivation

• Classification: predicting the class of a set of examples given their features.

• Standard classification methods aim at minimizing the errors.

• Such a traditional framework assumes that all misclassification errors carry the same cost.

• This is not the case in many real-world applications: Credit card fraud detection, churn modeling, credit scoring, direct marketing.
Agenda

• Cost-sensitive classification
  Background, previous contributions

• Cost-sensitive Ensembles
  Introduction, random inducers, combination methods, propose algorithms

• Datasets
  Credit card fraud detection, churn modeling, credit scoring, direct marketing

• Experiments
  Experimental setup, results

• Conclusions
  Contributions
Background - Binary classification

**predict the class** of set of examples given their features

\[ f : \mathcal{S} \rightarrow \{0, 1\} \]

Where each element of \( \mathcal{S} \) is composed by \( \mathbf{X}_i = [x^1_i, x^2_i, \ldots, x^k_i]\)

It is usually evaluated using a traditional misclassification measure such as Accuracy, F1Score, AUC, among others.

However, these measures assumes that different misclassification errors carry the **same cost**
We define a cost measure based on the **cost matrix** [Elkan 2001]

<table>
<thead>
<tr>
<th></th>
<th>Actual Positive $y_i = 1$</th>
<th>Actual Negative $y_i = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predicted Positive</strong> $c_i = 1$</td>
<td>$C_{TP_i}$</td>
<td>$C_{FP_i}$</td>
</tr>
<tr>
<td><strong>Predicted Negative</strong> $c_i = 0$</td>
<td>$C_{FN_i}$</td>
<td>$C_{TN_i}$</td>
</tr>
</tbody>
</table>

From which we calculate the **cost** of applying a classifier to a given set

$$Cost(f(S)) = \sum_{i=1}^{N} y_i (c_i C_{TP_i} + (1 - c_i) C_{FN_i}) + (1 - y_i) (c_i C_{FP_i} + (1 - c_i) C_{TN_i})$$
Background - Cost-sensitive evaluation

However, the total cost may not be easy to interpret. Therefore, we propose a savings measure as the cost vs. the cost of using no algorithm at all.

\[
Savings(f(S)) = \frac{Cost_l(S) - Cost(f(S))}{Cost_l(S)}
\]

Where \( Cost_l(S) \) is the cost of predicting the costless class.

\[
Cost_l(S) = \min\{Cost(f_0(S)), Cost(f_1(S))\}
\]
Background - State-of-the-art methods

Research in example-dependent cost-sensitive classification has been narrow, mostly because of the lack of publicly available datasets [Aodha and Brostow 2013].

Standard approaches consist in re-weighting the training examples based on their costs:

• Cost-proportionate rejection sampling [Zadrozny et al. 2003]

• Cost-proportionate oversampling [Elkan 2001]
Previous contributions

• **Bayes minimum risk**

• **Probability calibration for Bayes minimum risk (BMR)**

• **Cost-sensitive logistic regression (CSLR)**

• **Cost-sensitive decision trees (CSDT)**
Cost-sensitive classification
  Background, previous contributions

Cost-sensitive Ensembles
  Introduction, random inducers, combination methods, propose algorithms

Datasets
  Credit card fraud detection, churn modeling, credit scoring, direct marketing

Experiments
  Experimental setup, results

Conclusions
  Contributions
The main idea behind the ensemble methodology is to **combine several individual base classifiers** in order to have a classifier that outperforms everyone of them.

### “The Blind Men and the Elephant”, Godfrey Saxe’s

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
<th>Model 4</th>
<th>Model 5</th>
<th>Model 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some unknown distribution
Typical ensemble is made by combining $T$ different base classifiers. Each base classifier is trained by applying algorithm $M$ in a random subset

$$M_j = M(S_j) \quad \forall j \in \{1..T\}$$
Random inducers

Training set

Bagging

Pasting

Random forest

Random patches

12
Proposed combination methods

After the base classifiers are constructed they are typically combined using one of the following methods:

- **Majority voting**

\[
H(S) = f_{mv}(S, M) = \arg \max_{c \in \{0,1\}} \sum_{j=1}^{T} 1_c(M_j(S))
\]

- **Proposed cost-sensitive weighted voting**

\[
H(S) = f_{wv}(S, M, \alpha) = \arg \max_{c \in \{0,1\}} \sum_{j=1}^{T} \alpha_j 1_c(M_j(S))
\]

\[
\alpha_j = \frac{1 - \epsilon(M_j(S_j^{oob}))}{\sum_{j_1=1}^{T} 1 - \epsilon(M_{j_1}(S_{j_1}^{oob}))}
\]

\[
\alpha_j = \frac{\text{Savings}(M_j(S_j^{oob}))}{\sum_{j_1=1}^{T} \text{Savings}(M_{j_1}(S_{j_1}^{oob}))}
\]

\[
S_j^{oob} = S - S_j
\]
Proposed combination methods

- Proposed cost-sensitive stacking

\[
H(S) = f_s(S, M, \beta) = \frac{1}{1 + e^{-(\sum_{j=1}^{T} \beta_j M_j(S))}}
\]

Using the cost-sensitive logistic regression [Correa et. al, 2014] model:

\[
J(S, M, \beta) = \sum_{i=1}^{N} \left[ y_i \left( f_s(X_i, M, \beta) \cdot (C_{TP_i} - C_{FN_i}) + C_{FN_i} \right) + (1 - y_i) \left( f_s(X_i, M, \beta) \cdot (C_{FP_i} - C_{TN_i}) + C_{TN_i} \right) \right]
\]

Then the weights are estimated using

\[
\beta = \arg \min_{\beta \in \mathbb{R}^T} J(S, M, \beta)
\]
The subsampling can be done either by: Bagging, pasting, random forest or random patches.
Agenda

• Cost-sensitive classification
  Background, previous contributions

• Cost-sensitive Ensembles
  Introduction, random inducers, combination methods, propose algorithms

• Datasets
  Credit card fraud detection, churn modeling, credit scoring, direct marketing

• Experiments
  Experimental setup, results

• Conclusions
  Contributions
Credit card fraud detection

Cost matrix

<table>
<thead>
<tr>
<th></th>
<th>Actual Positive</th>
<th>Actual Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Positive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_i = 1$</td>
<td>$C_a$</td>
<td>$C_a$</td>
</tr>
<tr>
<td>Predicted Negative</td>
<td>$Amt_i$</td>
<td>0</td>
</tr>
<tr>
<td>$c_i = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th># Examples</th>
<th>% Positives</th>
<th>Cost (Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,638,772</td>
<td>0.21%</td>
<td>860,448</td>
</tr>
</tbody>
</table>

Churn modeling

Cost matrix

<table>
<thead>
<tr>
<th></th>
<th>Actual Positive ( y_i = 1 )</th>
<th>Actual Negative ( y_i = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Pos</td>
<td>( C_{TP_i} = \gamma_i C_{oi} + (1 - \gamma_i)(CLV_i + C_a) )</td>
<td>( C_{FP_i} = C_{oi} + C_a )</td>
</tr>
<tr>
<td>( c_i = 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicted Neg</td>
<td>( C_{FN_i} = CLV_i )</td>
<td>( C_{TN_i} = 0 )</td>
</tr>
<tr>
<td>( c_i = 0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th># Examples</th>
<th>% Positives</th>
<th>Cost (Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9,410</td>
<td>4.83%</td>
<td>580,884</td>
</tr>
</tbody>
</table>

Credit scoring

Cost matrix

<table>
<thead>
<tr>
<th>Predicted Positive</th>
<th>Actual Positive $y_i = 1$</th>
<th>Actual Negative $y_i = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i = 1$</td>
<td>0</td>
<td>$r_i + C_{FP}^a$</td>
</tr>
<tr>
<td>$c_i = 0$</td>
<td>$C_l_i \cdot L_{gd}$</td>
<td>0</td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th></th>
<th># Examples</th>
<th>% Positives</th>
<th>Cost (Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kaggle Credit</td>
<td>112,915</td>
<td>6.74%</td>
<td>83,740,181</td>
</tr>
<tr>
<td>PAKDD09 Credit</td>
<td>38,969</td>
<td>19.88%</td>
<td>3,117,960</td>
</tr>
</tbody>
</table>

Direct marketing

Cost matrix

<table>
<thead>
<tr>
<th>Predicted Positive $c_i = 1$</th>
<th>Actual Positive $y_i = 1$</th>
<th>Actual Negative $y_i = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$C_a$</td>
<td>$C_a$</td>
</tr>
<tr>
<td>Predicted Negative $c_i = 0$</td>
<td>$Int_i$</td>
<td>0</td>
</tr>
</tbody>
</table>

Database

<table>
<thead>
<tr>
<th># Examples</th>
<th>% Positives</th>
<th>Cost (Euros)</th>
</tr>
</thead>
<tbody>
<tr>
<td>37,931</td>
<td>12.62%</td>
<td>59,507</td>
</tr>
</tbody>
</table>

Agenda

• Cost-sensitive classification
  Background, previous contributions

• Cost-sensitive Ensembles
  Introduction, random inducers, combination methods, propose algorithms

• Datasets
  Credit card fraud detection, churn modeling, credit scoring, direct marketing

• Experiments
  Experimental setup, results

• Conclusions
  Contributions
Experimental setup - Methods

- **Cost-insensitive (CI):**
  - Decision trees (DT)
  - Logistic regression (LR)
  - Random forest (RF)
  - Under-sampling (u)
- **Cost-proportionate sampling (CPS):**
  - Cost-proportionate rejection-sampling (r)
  - Cost-proportionate over-sampling (o)
- **Bayes minimum risk (BMR)**
- **Cost-sensitive training (CST):**
  - Cost-sensitive logistic regression (CSLR)
  - Cost-sensitive decision trees (CSDT)
Experimental setup - Methods

• **Ensemble cost-sensitive decision trees (ECSDT):**

**Random inducers:**
- Bagging (CSB)
- Pasting (CSP)
- Random forest (CSRF)
- Random patches (CSRP)

**Combination:**
- Majority voting (mv)
- Cost-sensitive weighted voting (wv)
- Cost-sensitive staking (s)
Experimental setup

- Each experiment was carried out 50 times.
- For the parameters of the algorithms, a grid search was made.
- Results are measured by savings.
- Then the Friedman ranking is calculated for each method.
## Results

Results of the **Friedman rank** of the savings (1=best, 28=worst)

<table>
<thead>
<tr>
<th>Family</th>
<th>Algorithm</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECSDT</td>
<td>CSRP-wv-t</td>
<td>2.6</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSRP-s-t</td>
<td>3.4</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSRP-mv-t</td>
<td>4</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSB-wv-t</td>
<td>5.6</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSP-wv-t</td>
<td>7.4</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSB-mv-t</td>
<td>8.2</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSRF-wv-t</td>
<td>9.4</td>
</tr>
<tr>
<td>BMR</td>
<td>RF-t-BMR</td>
<td>9.4</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSP-s-t</td>
<td>9.6</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSP-mv-t</td>
<td>10.2</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSB-s-t</td>
<td>10.2</td>
</tr>
<tr>
<td>BMR</td>
<td>LR-t-BMR</td>
<td>11.2</td>
</tr>
<tr>
<td>CPS</td>
<td>RF-r</td>
<td>11.6</td>
</tr>
<tr>
<td>CST</td>
<td>CSDT-t</td>
<td>12.6</td>
</tr>
<tr>
<td>CST</td>
<td>CSLR-t</td>
<td>14.4</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSRF-mv-t</td>
<td>15.2</td>
</tr>
<tr>
<td>ECSDT</td>
<td>CSRF-s-t</td>
<td>16</td>
</tr>
<tr>
<td>CI</td>
<td>RF-u</td>
<td>17.2</td>
</tr>
<tr>
<td>CPS</td>
<td>LR-r</td>
<td>19</td>
</tr>
<tr>
<td>BMR</td>
<td>DT-t-BMR</td>
<td>19</td>
</tr>
<tr>
<td>CPS</td>
<td>LR-o</td>
<td>21</td>
</tr>
<tr>
<td>CPS</td>
<td>DT-r</td>
<td>22.6</td>
</tr>
<tr>
<td>CI</td>
<td>LR-u</td>
<td>22.8</td>
</tr>
<tr>
<td>CPS</td>
<td>RF-o</td>
<td>22.8</td>
</tr>
<tr>
<td>CI</td>
<td>DT-u</td>
<td>24.4</td>
</tr>
<tr>
<td>CPS</td>
<td>DT-o</td>
<td>25</td>
</tr>
<tr>
<td>CI</td>
<td>DT-t</td>
<td>26</td>
</tr>
<tr>
<td>CI</td>
<td>RF-t</td>
<td>26.2</td>
</tr>
</tbody>
</table>
Results

Results of the **Friedman rank** of the savings organized by family

![Box plot showing Friedman ranking](image-url)
## Results

<table>
<thead>
<tr>
<th>Database</th>
<th>Algorithm</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraud</td>
<td>CSRP-wv-t</td>
<td>0.73</td>
</tr>
<tr>
<td>Churn</td>
<td>CSRP-s-t</td>
<td>0.17</td>
</tr>
<tr>
<td>Credit1</td>
<td>CSRP-mv-t</td>
<td>0.52</td>
</tr>
<tr>
<td>Credit2</td>
<td>LR-t-BMR</td>
<td>0.31</td>
</tr>
<tr>
<td>Marketing</td>
<td>LR-t-BMR</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Percentage of the highest savings**

![Graph showing percentage of the highest savings for different datasets and algorithms. The x-axis represents different databases (Fraud, Churn, Credit1, Credit2, Marketing) and the y-axis shows the percentage of the best model. The graph includes multiple lines for different algorithms: LR-t-BMR, CSDT-t, RF-t-BMR, and CSRP-wv-t.]
Results within the ECSDT family

By random inducer

By combination method
Conclusions

• New framework for ensembles of example dependent cost-sensitive decision trees

• Using five databases, from four real-world applications: credit card fraud detection, churn modeling, credit scoring and direct marketing, we show that the proposed algorithm significantly outperforms the state-of-the-art cost-insensitive and example-dependent cost-sensitive algorithms

• Highlight the importance of using the real example-dependent financial costs associated with the real-world applications
CostCla is a Python module for cost-sensitive machine learning built on top of Scikit-Learn, SciPy and distributed under the 3-Clause BSD license.

In particular, it provides:
• A set of example-dependent cost-sensitive algorithms
• Different real-world example-dependent cost-sensitive datasets.

Installation

pip install costcla

Documentation: https://pythonhosted.org/costcla/
Development: https://github.com/albahnsen/CostSensitiveClassification
Costcl - Software

Prepare dataset and load libraries

```python
In [38]:
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
from costcl.metrics import savings_score
from costcl.datasets import load_creditscoring2
from costcl.sampling import cost_sampling
from costcl import models
data = load_creditscoring2()
X_train, X_test, y_train, y_test,
cost_mat_train, cost_mat_test = \ntrain_test_split(data.data, data.target, data.cost_mat)
```

Random forest

```python
In [19]:
f_RF = RandomForestClassifier()
y_pred = f_RF.fit(X_train, y_train).predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)
0.042197359989
```

cost-proportonate rejection sampling

```python
In [26]:
X_cps_r, y_cps_r, cost_mat_cps_r = \ncost_sampling(X_train, y_train, cost_mat_train,
method='RejectionSampling')
y_pred = f_RF.fit(X_cps_r, y_cps_r).predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)
0.286743761779
```

Bayes minimum risk

```python
In [38]:
f_RF.fit(X_train, y_train)
y_prob_test = f_RF.predict_proba(X_test)
f_BMR = models.BayesMinimumRiskClassifier()
f_BMR.fit(y_test, y_prob_test)
y_pred = f_BMR.predict(y_prob_test, cost_mat_test)
print savings_score(y_test, y_pred, cost_mat_test)
0.285102564249
```

cost-sensitive decision tree

```python
In [2]:
f_CSDT = models.CSDDecisionTreeClassifier()
f_CSDT.fit(data.data, data.target, data.cost_mat)
y_pred = f_CSDT.predict(data.data)
print savings_score(data.target, y_pred, data.cost_mat)
0.289489571352
```

cost-sensitive random patches

```python
In [33]:
f_CSRP = costcl.models.CSRandomPatchesClassifier()
f_CSRP.fit(data.data, data.target, data.cost_mat)
y_pred = f_CSRP.predict(data.data)
print savings_score(data.target, y_pred, data.cost_mat)
0.306607400467
```
Costcla - Software


```
class costcla.models.CostSensitiveLogisticRegression(C=1.0, fit_intercept=True, max_iter=100, random_state=None, solver='ga', tol=0.0001, verbose=0)

A example-dependent cost-sensitive Logistic Regression classifier.

Parameters:

- **C**: float, optional (default=1.0)
  Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

- **fit_intercept**: bool, default: True
  Specifies if a constant (a.k.a. bias or intercept) should be added the decision function.

- **max_iter**: int
  Useful only for the ga and bfgs solvers. Maximum number of iterations taken for the solvers to converge.

- **random_state**: int seed, RandomState instance, or None (default)
  The seed of the pseudo random number generator to use when shuffling the data.

- **solver**: {'ga', 'bfgs'}
  Algorithm to use in the optimization problem.
```
Thank You!!

Alejandro Correa Bahnsen